Data Science and Engineering (2020) 5:168–179
https://doi.org/10.1007/s41019-020-00120-6

Space-Efficient Feature Maps for String Alignment Kernels

Yasuo Tabei[1] · **Yoshihiro Yamanishi**[2] · **Rasmus Pagh**[3]

Received: 4 February 2020 / Revised: 6 April 2020 / Accepted: 21 April 2020 / Published online: 18 May 2020

Abstract
String kernels are attractive data analysis tools for analyzing string data. Among them, alignment kernels are known for their high prediction accuracies in string classifications when tested in combination with SVM in various applications. However, alignment kernels have a crucial drawback in that they scale poorly due to their quadratic computation complexity in the number of input strings, which limits large-scale applications in practice. We address this need by presenting the first approximation for string alignment kernels, which we call *space-efficient feature maps for edit distance with moves (SFMEDM)*, by leveraging a metric embedding named *edit-sensitive parsing* and *feature maps (FMs)* of *random Fourier features (RFFs)* for large-scale string analyses. The original FMs for RFFs consume a huge amount of memory proportional to the dimension d of input vectors and the dimension D of output vectors, which prohibits its large-scale applications. We present novel *space-efficient feature maps (SFMs)* of RFFs for a space reduction from $O(dD)$ of the original FMs to $O(d)$ of SFMs with a theoretical guarantee with respect to concentration bounds. We experimentally test SFMEDM on its ability to learn SVM for large-scale string classifications with various massive string data, and we demonstrate the superior performance of SFMEDM with respect to prediction accuracy, scalability and computation efficiency.

Keywords Feature maps · Kernel approximation · String alignment kernels

1 Introduction

Massive string data are now ubiquitous throughout research and industry, in areas such as biology, chemistry, natural language processing and data science. For example, e-commerce companies face a serious problem in analyzing huge datasets of user reviews, question answers and purchasing histories [11, 23]. In biology, homology detection from huge collections of protein and DNA sequences is an important part for their functional analyses [26]. There is therefore a strong need to develop powerful methods to make best use of massive string data on a large scale.

Kernel methods [12] are attractive data analysis tools because they can approximate any (possibly nonlinear) function or decision boundary well with enough training data. In kernel methods, a kernel matrix a.k.a. Gram matrix is computed from training data and *nonlinear support vector machines (SVM)* are trained on the matrix. Although it is known that kernel methods achieve high prediction accuracy for various tasks such as classification and regression, they scale poorly due to a quadratic complexity in the number of training data [9, 13]. In addition, calculation of a classification requires, in the worst case, linear time in the number of training data, which limits large-scale applications of kernel methods in practice.

String kernels [10] are kernel functions that operate on strings, and a variety of string kernels using string similarity measures have been proposed [5, 18, 21, 26]. As state-of-the-art string kernels, string alignment kernels are known for high prediction accuracy in string classifications, such as remote homology detection for protein sequences [26] and time-series classifications [5, 36], when tested in combination with SVM. However, alignment kernels have a crucial drawback; that is, as in other kernel methods, they scale

✉ Yasuo Tabei
yasuo.tabei@riken.jp

Yoshihiro Yamanishi
yamani@bio.kyutech.ac.jp

Rasmus Pagh
pagh@itu.dk

[1] RIKEN Center for Advanced Intelligence Project, Tokyo, Japan

[2] Kyushu Institute of Technology, Fukuoka, Japan

[3] BARC, IT University of Copenhagen, Copenhagen, Denmark

poorly due to their quadratic computation complexity in the number of training data.

Kernel approximations using *feature maps (FMs)* have been proposed to solve the scalability issues regarding kernel methods. FMs project training data into low-dimensional vectors such that the kernel value (similarity) between each pair of training data is approximately equal to the inner product of the corresponding pair of low-dimensional vectors. Then, *linear SVM* are trained on the projected vectors, thereby significantly improving the scalability, while preserving their prediction accuracy. Although a variety of kernel approximations using FMs for enhancing the scalability of kernel methods have been proposed (e.g., Jaccard kernels [20], polynomial kernels [24] and min–max kernels [19]), and *random Fourier features (RFFs)* [25] are an approximation of shift-invariant kernels (e.g., Laplacian and radial basis function (RBF) kernels), approximation for string alignment kernels has not been studied. Thus, an important open challenge, which is required for large-scale analyses of string data, is to develop a kernel approximation for string alignment kernels.

Several metric embeddings for string distance measures have been proposed for large-scale string processing [2, 4]. *Edit-sensitive parsing (ESP)* [4] is a metric embedding of a string distance measure called *edit distance with moves (EDM)* that consists of ordinal edit operations of insertion, deletion and replacement in addition to substring move operation. ESP maps all the strings from the EDM space into integer vectors named *characteristic vectors* in the L_1 distance space.

To date, ESP has been applied only to string processing such as string compression [22], indexing [31] and edit distance computation [4]; however, as we will see, there remains high potential for application to an approximation of alignment kernels. ESP is expected to be effective for approximating alignment kernels, because it approximates EDM between strings as L_1 distance between integer vectors.

Contribution In this paper, we present SFMEDM as the first approximation of alignment kernels for solving large-scale learning problems on string data. Key ideas behind the proposed method are threefold: (1) to project input strings into characteristic vectors leveraging ESP, (2) to map characteristic vectors into vectors of RFFs by FMs and (3) to train linear SVM on the mapped vectors. However, applying FMs for RFFs to high-dimensional vectors in a direct way requires memory linearly proportional to not only dimension d of input vectors but also dimension D of RFF vectors. In fact, characteristic vectors as input vectors for FMs tend to be very high-dimensional d for solving large-scale problems using FMs, and output vectors of RFFs need to also be high-dimensional D for achieving high prediction accuracies, and those conditions limit the applicability of FMs on a large scale. Although fastfood approach [17] and orthogonal range reporting [34] have been proposed for efficiently computing RFFs in $O(D \log d)$ time and $O(d)$ memory, they are only applicable to RFFs for approximating RBF kernels with a theoretical guarantee. Accordingly, in this study, we present *space-efficient FMs (SFMs)* that require only $O(d)$ memory to solve this problem and can be used for approximating any shift-invariant kernel such as a Laplacian kernel. This is an essential property which is required for approximating alignment kernels and has not been taken into account by previous research. Our SFMEDM has the following desirable properties:

(1) *Scalability* SFMEDM is applicable to massive string data.
(2) *Fast Training* SFMEDM trains SVM fast.
(3) *Space Efficiency* SFMEDM trains SVM space-efficiently.
(4) *Prediction Accuracy* SFMEDM can achieve high prediction accuracy.

We experimentally test the ability of SFMEDM to train SVM with various massive string data and demonstrate that SFMEDM has superior performance in terms of prediction accuracy, scalability and computational efficiency.

2 Related Work

String kernel is a kernel method for analyzing string data, and many string kernels have been proposed thus far [8, 16, 29]. Among them, string alignment kernels are known for more powerful and efficient. We briefly review the state of the art, which is also summarized in Table 1. Early methods are proposed in [1, 28, 36] and are known not to satisfy the positive definiteness for their kernel matrices. Thus, they are

Table 1 Summary of string alignment kernels

	Approach	Training time	Training space	Prediction time
GAK [5, 6]	Global alignment	$O(N^2L^2)$	$O(N^2)$	$O(NL^2)$
LAK [26]	Local alignment	$O(N^2L^2)$	$O(N^2)$	$O(NL^2)$
D2KE [32, 33]	Random feature map	$O(NDL^2)$	$O(N(L+D))$	$O(DL^2)$
SFMEDM (this study)	ESP	$O(NL + dDN)$	$O(NL \log NL + ND + d)$	$O(L + dD)$
SFMCGK (this study)	CGK	$O(NL + dDN)$	$O(L\lvert\Sigma\rvert + ND + d)$	$O(L + dD)$

proposed with numerical corrections for any deficiency of the kernel matrices.

The *global alignment kernel (GAK)* [5, 6] is an alignment kernel based on global alignments originally proposed for time-series data. GAK defines a kernel as a summation score of all possible global alignments between two strings. The computation time of GAK is $O(N^2L^2)$ for number of strings N and the length of strings L, and its space usage is $O(N^2)$.

A *local alignment kernel (LAK)* on the notion of the Smith–Waterman algorithm [30] for detecting protein remote homology was proposed by Saigo et al. [26]. LAK measures the similarity between each pair of strings by summing up scores obtained from local alignments with gaps of strings. The computation time of LAK is $O(N^2L^2)$ and its space usage is $O(N^2)$. Although, in combination with SVM, LAK achieves high classification accuracies for protein sequences, LAK is applicable to protein strings only because its scoring function is optimized for proteins.

D2KE [32] is a random feature map from structured data to feature vectors such that a distance measure between each pair of the structured data is preserved by the inner product between the corresponding pair of mapped vectors. The feature vector for each input structured data is built as follows: (1) D structured data in input are sampled; (2) the D-dimensional feature vector for each of the structured data is built such that each dimension of the feature vector is defined as the distance between the structured data and a sampled one. D2KE has been applied to time-series data [33]; however, as we will see, D2KE cannot achieve high prediction accuracies when it is applied to string data.

Despite the importance of a scalable learning with alignment kernels, no previous work has been able to achieve high scalabilities while preserving high prediction accuracies. We present SFMEDM, the first scalable learning with string alignment kernels that meets these demands and is made possible by leveraging an idea behind ESP and SFM.

CGK [2] is another metric embedding for edit distance and maps input strings S_i of alphabet Σ and of the maximum length L into strings S'_i of fixed-length L such that the edit distance between each pair of input strings is approximately preserved by the Hamming distance between the corresponding pair of mapped strings. Recently, CGK has been applied to the problem of edit similarity joins [35]. We also present a kernel approximation of alignment kernels called *SFMCGK* by leveraging an idea behind CGK and SFM.

Details of the proposed method are presented in the next section.

3 Edit-Sensitive Parsing

Edit-sensitive parsing (ESP) [4] is an approximation method for efficiently computing *edit distance with moves (EDM)*. EDM is a string-to-string distance measure for turning one string into another in a series of string operations, where a substring move is included as a string operation in addition to typical string operations such as insertion, deletion and replacement. Formally, let S be a string of length L and $S[i]$ be the ith character in S. $EDM(S, S')$ for two strings S and S' is defined as the minimum number of edit operations defined below to transform S into S' as following:

Insertion Character a at position i in S is inserted, resulting in $S[1] \dots S[i-1]aS[i]S[i+1] \dots S[L]$;

Deletion Character $S[i]$ at position i in S is deleted, resulting in $S[1] \dots S[i-1]S[i+1] \dots S[L]$;

Replacement Character $S[i]$ at position i in S is replaced by a, resulting in $S[1] \dots S[i-1]aS[i+1] \dots S[L]$;

Substring Move A substring $S[i]S[i+1] \dots S[j]$ in S is moved and inserting at position p, resulted in $S[1] \dots S[i-1]S[j+1] \dots S[p-1]S[i] \dots S[j]S[p] \dots S[L]$.

Computing EDM between two strings is known as an NP-complete problem [27]. ESP can approximately compute EDM by embedding strings into L_1 vector space by a parsing.

Given string S, ESP builds a parse tree named an *ESP tree*, which is illustrated in Fig. 1 as an example. The ESP tree is a balanced tree, and each node in the ESP tree belongs to one of three types: (1) a node with three children, (2) a node with two children and (3) a node without children (i.e., a leaf). In addition, internal nodes in the ESP tree have the same node label if and only if they have children satisfying both two conditions: (1) the numbers of those children are the same and (2) the node labels of those children are the same in the left-to-right order. The height of ESP tree is $O(\log L)$ for the length of input string L.

Let $V(S) \in \mathbf{N}^d$ be a d-dimensional integer vector built from ESP tree $T(S)$ such that each dimension of $V(S)$ is the number of a node label appearing in T. $V(S)$ is called *characteristic vectors*. ESP builds ESP trees such that as many subtrees with the same node labels as possible are built for common substrings for strings S and S', resulted

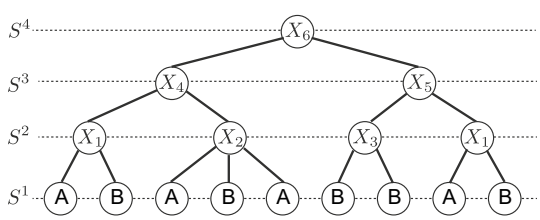

Fig. 1 Illustration of an ESP tree for string $S = ABABABBAB$

in an approximation of EDM between S and S' by L_1 distance between their characteristic vectors $V(S)$ and $V(S')$, i.e., $EDM(S,S') \approx ||V(S) - V(S')||_1$, where $||\cdot||_1$ is an L_1 norm. More precisely, the upper and lower bounds of the approximation are as follows:

$$\begin{aligned} EDM(S,S') &\leq ||V(S) - V(S')||_1 \\ &\leq O(\log L \log^* L) EDM(S,S'), \end{aligned} \tag{1}$$

where $\log^* L$ is the iterated logarithm of L, which is recursively defined as $\log^1 L = \log_2 L$, $\log^{i+1} L = \log\log^i L$ and $\log^* L = \min\{k; \log^k L \leq 1, i \geq 1\}$ for a positive integer L.

Detail of the ESP algorithm is presented in "Appendix."

4 Space-Efficient Feature Maps

In this section, we present our new SFMs for RFFs using space proportional to the dimension d of characteristic vectors and *independent* of the RFF target dimension D. The proposed SFMs improve space usage for generating RFFs from $O(dD)$ to $O(d)$ while preserving theoretical guarantees (concentration bounds). The method is general and can be used for approximating any shift-invariant kernel.

From an abstract point of view, an RFF is based on a way of constructing a random mapping

$$z_{\mathbf{r}} : \mathbf{R}^d \to [-1,+1]^2$$

such that for every choice of vectors $\mathbf{x}, \mathbf{y} \in \mathbf{R}^d$ we have

$$\mathbf{E}[z_{\mathbf{r}}(\mathbf{x})' z_{\mathbf{r}}(\mathbf{y})] = k(\mathbf{x},\mathbf{y}),$$

where k is the kernel function. The randomness of $z_{\mathbf{r}}$ comes from a vector $\mathbf{r} \in \mathbf{R}^d$ sampled from an appropriate distribution $\mathcal{D}_k$ that depends on kernel function k (see Sect. 5 for more details), and the expectation is over the choice of $\mathbf{r}$. For the purposes of this section, all that needs to be known about $\mathcal{D}_k$ is that the d vector coordinates are independently sampled according to the marginal distribution Δ_k.

Since $(z_{\mathbf{r}}(\mathbf{x})' z_{\mathbf{r}}(\mathbf{y}))^2 \leq 1$, we have $Var(z_{\mathbf{r}}(\mathbf{x})' z_{\mathbf{r}}(\mathbf{y})) \leq 1$, i.e., bounded variance; however, this in itself does not imply the desired approximation as $k(\mathbf{x},\mathbf{y}) \approx z_{\mathbf{r}}(\mathbf{x})' z_{\mathbf{r}}(\mathbf{y})$. Indeed, $z_{\mathbf{r}}(\mathbf{x})' z_{\mathbf{r}}(\mathbf{y})$ is a poor estimator of $k(\mathbf{x},\mathbf{y})$. The accuracy of RFFs can be improved by increasing the output dimension to $D \geq 2$. Specifically, RFFs use $D/2$ *independent* vectors $\mathbf{r}_1, \ldots, \mathbf{r}_{D/2} \in \mathbf{R}^d$ sampled from $\mathcal{D}_k$, and they consider FMs

$$\mathbf{z} : \mathbf{x} \mapsto \sqrt{\frac{2}{D}} \left(z_{\mathbf{r}_1}(\mathbf{x}), z_{\mathbf{r}_2}(\mathbf{x}), \ldots, z_{\mathbf{r}_{D/2}}(\mathbf{x}) \right)$$

that concatenates the values of $D/2$ functions to one D-dimensional vector. It can then be shown that $|\mathbf{z}(\mathbf{x})'\mathbf{z}(\mathbf{y}) - k(\mathbf{x},\mathbf{y})| \leq \varepsilon$ with high probability for sufficiently large $D = \Omega(1/\varepsilon^2)$.

To represent the function $\mathbf{z}$, it is necessary to store a matrix containing vectors $\mathbf{r}_1, \ldots, \mathbf{r}_{D/2}$, which uses space $O(dD)$. Our assumption for ensuring good kernel approximations is that the vectors $\mathbf{r}_i$ do not need to be independent. Instead, for a small integer parameter $t \in N$, we compute each vector $\mathbf{r}_i$ using a hash function $h : \{1, \ldots, D/2\} \to \mathbf{R}^d$ chosen from a t-wise independent family such that for every i, $h(i)$ comes from distribution $\mathcal{D}_k$. Then, instead of storing $\mathbf{r}_1, \ldots, \mathbf{r}_{D/2}$, we only store the description of the hash function h in memory $O(td)$. A priori, two issues seemingly concern this approach:

- It is unclear how to construct t-wise independent hash functions with output distribution $\mathcal{D}_k$.
- Is t-wise independence sufficient to ensure results similar to the fully independent setting?

We address these issues in the next two subsections.

4.1 Hash Functions with Distribution $\mathcal{D}_k$

For concreteness, our construction is based on the following class of t-wise independent hash functions, where $t \in \mathbf{N}$ is a parameter: For $\mathbf{a} = (a_0, a_1, \ldots, a_{t-1}) \in [0,1]^t$ chosen uniformly at random, let

$$f_{\mathbf{a}}(x) = \sum_{j=0}^{t-1} a_j x^j \bmod 1,$$

where $y \bmod 1$ computes the fractional part of $y \in \mathbf{R}$. It can be shown that any t distinct integer inputs $i_1, \ldots, i_t \in \mathbf{N}$, the vector $(f_{\mathbf{a}}(i_1), \ldots, f_{\mathbf{a}}(i_t))$ is uniformly distributed in $[0,1]^t$.

Let CDF^{-1} denote the inverse of the cumulative distribution function of the marginal distribution Δ_k. Then, if y is uniformly distributed in $[0,1]$, $CDF^{-1}(y) \sim \Delta_k$. Accordingly, hash function h can be constructed where the jth coordinate on input i is given, as

$$h(i)_j = CDF^{-1}(f_{\mathbf{a}^j}(i)),$$

where $\mathbf{a}^1, \ldots, \mathbf{a}^d$ are chosen independently from $[0,1]^d$. We see that for every $i \in \mathbf{N}$, $h(i) = (h(i)_1, \ldots, h(i)_d)$ has distribution $\mathcal{D}_k$. Furthermore, for every set of t distinct integer inputs $i_1, \ldots, i_t \in \mathbf{N}$, the hash values $h(i_1), \ldots, h(i_t)$ are independent.

4.2 Concentration Bounds

We then show that for RFFs, $D = O(1/\varepsilon^2)$ random features suffice to approximate the kernel function within error ε with probability arbitrarily close to 1.

Theorem 1 *For every pair of vectors* $\mathbf{x}, \mathbf{y} \in \mathbf{R}^d$, *if the mapping* $\mathbf{z}$ *is constructed as described above using* $t \geq 2$, *for every* $\varepsilon > 0$, *it follows that*

$$\Pr[|\mathbf{z}(\mathbf{x})'\mathbf{z}(\mathbf{y}) - k(\mathbf{x}, \mathbf{y})| \geq \varepsilon] \leq 2/(\varepsilon^2 D).$$

Proof Our proof follows the same outline as the standard proof of Chebyshev's inequality. Consider the second central moment:

$$\begin{aligned}
&\mathbf{E}\left[\left(z(\mathbf{x})'z(\mathbf{y}) - k(\mathbf{x}, \mathbf{y})\right)^2\right] \\
&= \mathbf{E}\left[\left(\frac{2}{D}\sum_{i=1}^{D/2} z_{h(i)}(\mathbf{x})'z_{h(i)}(\mathbf{y}) - k(\mathbf{x}, \mathbf{y})\right)^2\right] \\
&= \mathbf{E}\left[\sum_{i=1}^{D/2}\left(z_{h(i)}(\mathbf{x})'z_{h(i)}(\mathbf{y}) - \frac{D}{2}k(\mathbf{x}, \mathbf{y})\right)^2\right] \\
&= \frac{4}{D^2}\sum_{i=1}^{D/2}\mathbf{E}\left[\left(z_{h(i)}(\mathbf{x})'z_{h(i)}(\mathbf{y}) - \frac{D}{2}k(\mathbf{x}, \mathbf{y})\right)^2\right] \leq 2/D.
\end{aligned}$$

The second equality above uses 2-wise independence, and the fact that

$$\mathbf{E}\left[\sum_{i=1}^{D/2} z_{h(i)}(\mathbf{x}) \cdot z_{h(i)}(\mathbf{y}) - \frac{D}{2}k(\mathbf{x}, \mathbf{y})\right] = 0$$

to conclude that only $D/2$ terms in the expansion have nonzero expectation. Finally, we have:

$$\begin{aligned}
&\Pr[|\mathbf{z}(\mathbf{x})'\mathbf{z}(\mathbf{y}) - k(\mathbf{x}, \mathbf{y})| \geq \varepsilon] \\
&\quad \leq \Pr[(\mathbf{z}(\mathbf{x})'\mathbf{z}(\mathbf{y}) - k(\mathbf{x}, \mathbf{y}))^2 \geq \varepsilon^2] \\
&\quad \leq \mathbf{E}[(\mathbf{z}(\mathbf{x})'\mathbf{z}(\mathbf{y}) - k(\mathbf{x}, \mathbf{y}))^2]/\varepsilon^2 \leq 2/(\varepsilon^2 D),
\end{aligned}$$

where the second inequality follows from Markov's inequality. This concludes the proof.

In the original analysis of RFFs, a strong approximation guarantee was considered; namely, the kernel function for *all* pairs of points $\mathbf{x}, \mathbf{y}$ in a bounded region of $\mathbf{R}^d$ was approximated. This kind of result can be achieved by choosing $t \geq 2$ sufficiently large to obtain strong tail bounds. However, we show that the point-wise guarantee (with $t = 2$) provided by Theorem 1 is sufficient for an application in kernel approximations in Sect. 7.

5 Scalable Alignment Kernels

We present the SFMEDM algorithm for scalable learning with alignment kernels hereafter. Let us assume a collection of N strings and their labels $(S_1, y_1), (S_2, y_2), \ldots, (S_N, y_N)$ where $y_i \in \{0, 1\}$. We define alignment kernels using $EDM(S_i, S_j)$ for each pair of strings S_i and S_j as follows:

$$k(S_i, S_j) = \exp(-EDM(S_i, S_j)/\beta), \tag{2}$$

where β is a parameter. We apply ESP to each S_i for $i = 1, 2, \ldots, N$ and build ESP trees $T_1, T_2, \ldots, T_N$. Since ESP approximates $EDM(S_i, S_j)$ as an L_1 distance between characteristic vectors $V(S_i)$ and $V(S_j)$ built from ESP trees T_i and T_j for S_i and S_j, i.e., $EDM(S_i, S_j) \approx ||V(S_i) - V(S_j)||_1$, $k(S_i, S_j)$ can be approximated as follows:

$$k(S_i, S_j) \approx \exp(-||V(S_i) - V(S_j)||_1/\beta). \tag{3}$$

Since Eq. 3 is a Laplacian kernel, which is also known as a shift-invariant kernel [25], we can approximate $k(S_i, S_j)$ using FMs $\mathbf{z}(\mathbf{x})$ for RFFs as follows:

$$k(S_i, S_j) \approx \mathbf{z}(V(S_i))'\mathbf{z}(V(S_j)), \tag{4}$$

where $\mathbf{z}(\mathbf{x}) = \sqrt{\frac{2}{D}}(\mathbf{z}_{\mathbf{r}_1}(\mathbf{x}), \mathbf{z}_{\mathbf{r}_2}(\mathbf{x}), \ldots, \mathbf{z}_{\mathbf{r}_{D/2}}(\mathbf{x}))$. For Laplacian kernels, $\mathbf{z}_{\mathbf{r}_m}(\mathbf{x})$ for each $m = 1, 2, \ldots, D/2$ is defined as

$$\mathbf{z}_{\mathbf{r}_m}(\mathbf{x}) = (\cos(\mathbf{r}_m^\top \mathbf{x}), \sin(\mathbf{r}_m^\top \mathbf{x})), \tag{5}$$

where random vectors $\mathbf{r}_m \in \mathbf{R}^d$ for $m = 1, 2, \ldots, D/2$ are sampled from the Cauchy distribution. We shall refer to approximations of alignment kernels leveraging ESP and FMs as *FMEDM*.

Algorithm 1 Generation of Cauchy random numbers using 2-wise independent hash function. $array_1$, $array_2$: arrays of d 64-bit unsigned integers; $UMAX32$: maximum value of unsigned 32-bit integer; β: a parameter.

1: Initialize $array_1$ and $array_2$ with 64-bit random numbers as unsigned integers.
2: **function** FUNC_F(i, j)
3: $f = array_1[j] + array_2[j] \cdot i$ ▷ Compute hash value
4: $v = f >> 32$ ▷ Get the most-significant 32-bit of value
5: **return** $v/UMAX32$ ▷ Return value in $[0, 1]$
6: **function** FUNC_H(i, j)
7: $u = Func\_F(i, j)$
8: **return** $tan(\pi \cdot (u - 0.5))/\beta$ ▷ Convert random number u to Cauchy random number

Algorithm 2 Construction of RFFs by SFMs. $\mathbf{z}$: vector of RFFs; D: dimension of $\mathbf{z}$; V: characteristic vector; d: dimension of V.

1: **function** SFM(V)
2: **for** $i = 1, ..., D/2$ **do**
3: $s = 0$
4: **for** $j = 1, ..., d$ **do**
5: $s = s + V[j] \cdot Func\_H(i, j)$
6: $\mathbf{z}[2 \cdot i - 1] = \sqrt{\frac{2}{D}} \cdot sin(s)$
7: $\mathbf{z}[2 \cdot i] = \sqrt{\frac{2}{D}} \cdot cos(s)$
8: **return** $\mathbf{z}$

Applying FMs to high-dimensional characteristic vectors consumes $O(dD)$ memory for storing vectors $\mathbf{r}_m \in \mathbf{R}^d$ for $m = 1, 2, \ldots, D/2$. Thus, we present SFMs for RFFs using only $O(td)$ memory by applying t-wise independent hash functions introduced in Sect. 4. We fix $t = 2$ in this study, resulted in $O(d)$ memory. We shall refer to approximations of alignment kernels leveraging ESP and SFMs as *SFMEDM*.

Algorithm 1 generates random numbers from a Cauchy distribution by using $O(d)$ memory. Two arrays $array_1$ and $array_2$, initialized with 64-bit random numbers as unsigned integers, are used. Function $f_a(x)$ is implemented using $array_1$ and $array_2$ in *Func_F* and returns a random number in [0, 1] for given i and j as input. Then, random number u returned from *Func_F* is converted to a random number from the Cauchy distribution in *Func_H* as $\tan(\pi \cdot (u - 0.5))/\beta$ at line 8. Algorithm 2 implements SFMs generating RFFs in Eq. 5. Computation time and memory for SFMs are $O(dDN)$ and $O(d)$, respectively.

6 Feature Maps Using CGK Embedding

CGK [2, 35] is another string embedding using a randomized algorithm. Let S_i for i = 1,2,⋯,N be input strings of alphabet Σ and let L be the maximum length of input strings. CGK maps input strings S_i in the edit distance space into strings S'_i of length L in the Hamming space, i.e., the edit distance between each pair S_i and S_j of input strings is approximately preserved by the Hamming distance of the corresponding pair S'_i and S'_j of the mapped strings. See [35] for the detail of CGK.

To apply SFMs, we convert mapped strings S'_i in the Hamming space by CGK to characteristic vectors $V^C(S'_i)$ in the L_1 distance space as follows. We view elements $S'_i[j]$ for j = 1,2, ..., L as locations (of the nonzero elements) instead of characters. For example, when $\Sigma = \{1, 2, 3\}$, we view each $S'_i[j]$ as a vector of length $|\Sigma| = 3$. If $S'_i[j] = 1$, then we code it as (0.5, 0, 0); if $S'_i[j] = 3$, then we code it as (0, 0, 0.5). We then concatenate those L vectors into one vector $V^C(S'_i)$ of dimension $L|\Sigma|$ and with L nonzero elements. As a result, the Hamming distance between original strings S'_i and S'_j is equal to the L_1 distance between obtained vectors $V^C(S'_i)$ and $V^C(S'_j)$, i.e., $Ham(S'_i, S'_j) = ||V^C(S'_i) - V^C(S'_j)||_1$. By applying SFMs or FMs to $V^C(S'_i)$, we built vectors of RFFs $\mathbf{z}(V^C(S'_i))$. We shall call approximations of alignment kernels using CGK and SFMs (respectively, FMs) *SFMCGK* (respectively, *FMCGK*).

7 Experiments

In this section, we evaluated the performance of SFMEDM with five massive string datasets, as shown in Table 2. The "Protein" and "DNA" datasets consist of 3238 human enzymes obtained from the KEGG GENES database [14], respectively. Each enzyme in "DNA" was coded by a string consisting of four types of nucleotides or bases (i.e., *A*, *T*, *G* and *C*). Similarity, each enzyme in "Protein" was coded by a string consisting of 20 types of amino acids. Enzymes belonging to the isomerases class in the enzyme commission (EC) numbers in "DNA" and "Protein" have positive labels and the other enzymes have negative labels. The "Music" and "Sports" datasets consist of 10,261 and 296,337 reviews of musical instruments products and sports products in English from Amazon [11, 23], respectively. Each review has a rating of five levels. We assigned positive labels to reviews with four or five levels for rating and negative labels to the other reviews. The "Compound" dataset consists of 1,367,074 bioactive compounds obtained from the NCBI PubChem database [15]. Each compound was coded by a string representation of chemical structures called *SMILES*. The biological activities of the compounds for human proteins were obtained from the ChEMBL database. In this study, we focused on the biological activity for the human protein microtubule-associated protein tau (MAPT). The label of each compound corresponds to the presence or absence of biological activity for MAPT.

All the methods were implemented by C++, and all the experiments were performed on one core of a quad-core Intel Xeon CPU E5-2680 (2.8 GHz). The execution of each method was stopped if it did not finish within 48 h in the

Table 2 Summary of datasets

Dataset	Number	# positives	Alphabet size	Average length
Protein	3238	96	20	607
DNA	3238	96	4	1827
Music	10,261	9022	61	329
Sports	296,337	253,017	63	307
Compound	1,367,074	57,536	44	53

experiments. Software and datasets used in this experiment are downloadable from https://sites.google.com/view/alignmentkernels/home.

7.1 Scalability of ESP

First, we evaluated the scalability of ESP and CGK. Table 3 shows the execution time, memory in megabytes and dimension d of characteristic vectors generated by ESP and CGK. ESP and CGK were practically fast enough to build characteristic vectors for large datasets. The executions of ESP and CGK finished within 60 s for "Compound" that was the largest dataset consisting of more than 1 million compounds. At most 1.5 GB memory was consumed in the execution of ESP. These results demonstrated high scalability of ESP for massive datasets.

For each dataset, characteristic vectors of very high dimensions were built by ESP and CGK. For example, 18 million dimension vectors were built by ESP for the "Sports" dataset. Applying the original FMs for RFFs to such high-dimensional characteristic vectors consumed huge amount of memory, deteriorating the scalability of FMs. The proposed SFMs can solve the scalability problem, which will be shown in the next subsection.

7.2 Efficiency of SFMs

We evaluated the efficiency of SFMs applied to characteristic vectors built from ESP, and we compared SFMs with FMs. We examined combinations of characteristic vectors and projected vectors of SFMEDM, FMEDM, SFMCGK and FMCGK. The dimension D of projected vectors of RFFs was examined for $D = \{128, 512, 2048, 8192, 16384\}$.

Figure 2 shows the amount of memory consumed in SFMs and FMs for characteristic vectors built by ESP and CGK for each dataset. According to the figure, a huge amount of memory was consumed by FMs for high-dimensional characteristic vectors and projected vectors. Around 1.1TB and 323 GB of memory were consumed by FMEDM for $D = 16{,}384$ for "Sports" and "Compound," respectively. Those huge amounts of memory made it impossible to build high-dimensional vectors of RFFs. The memory required by SFMs was linear in regard to dimension d of characteristic vectors for each dataset. Only 280 MB and 80 MB of memory were consumed by SFMEDM for $D = 16{,}384$ for "Sports" and "Compound," respectively. These results suggest that compared with FMEDM, SFMEDM dramatically reduces the amount of required memory.

Figure 3 shows the execution time for building projected vectors for each dataset. According to the figure, execution time increases linearly with dimension D for each method and for "Compound," SFMs built 16,384-dimensional vectors of RFFs in around 9 h.

We evaluated accuracies of our approximations of alignment kernels in terms of average error of RFFs, defined as

$$\sum_{i=1}^{N}\sum_{j=i}^{N} |k(S_i, S_j) - \mathbf{z}(V(S_i))'\mathbf{z}(V(S_j))|/(N(N+1)/2),$$

where $k(S_i, S_j)$ is defined by Eq. 3 and $\beta = 1$ was fixed. Average error of SFMs was compared with that of FMs for each dataset. Table 4 shows average error of SFMs and FMs using characteristic vectors built from ESP for each dataset. The average errors of SFMEDM and FMEDM are almost the same for all datasets and dimension D. The accuracies of FMs were preserved in the case of SFMs, while the amount

Table 3 Execution time in seconds, memory in mega bytes and dimension d of characteristic vectors by ESP and CGK for each dataset

Data	Protein		DNA		Music		Sports		Compound	
Method	ESP	CGK	ESP	CGK	ESP	CGK	ESP	CGK	ESP	CGK
Time (s)	1.25	0.87	2.01	2.63	1.86	0.86	47.83	34.08	32.73	28.70
Memory (MB)	1042.90	0.09	1049.89	3.38	1048.30	0.37	1514.23	0.54	1165.60	0.08
Dimension d	707, 922	4, 950, 686	969, 653	19, 192, 656	910, 110	2, 129, 505	18, 379, 173	3, 095, 844	5, 302, 660	485, 840

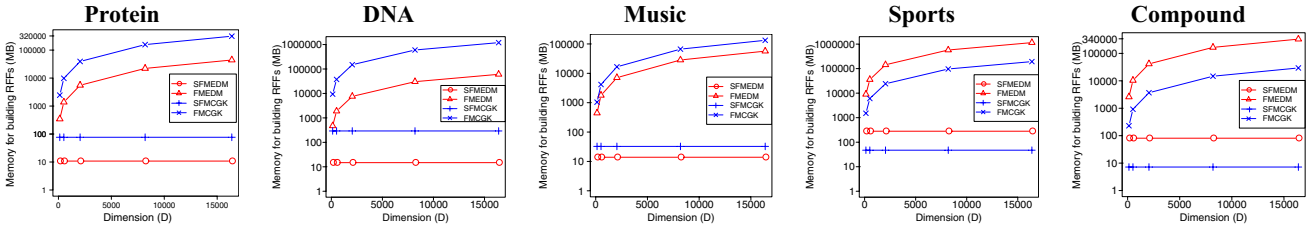

Fig. 2 Memory in megabytes for building vectors of RFFs for various dimensions D

of memory required by FMs was dramatically reduced. The same tendencies were observed for average errors of SFMs in combination with CGK, as shown in Table 5.

7.3 Classification Performance of SFMEDM

We evaluated classification abilities of SFMEDM, SFMCGK, D2KE, LAK and GAK. We used an implementation of LAK downloadable from http://sunflower.kuicr.kyoto-u.ac.jp/~hiroto/project/homology.html. We implemented D2KE by C++ with edit distance as a distance measure for strings. Laplacian kernels with characteristic vectors of ESP and CGK in Eq. 3 were also evaluated and denoted as ESPKernel and CGKKernel, respectively. In addition, we evaluated a classification ability of the state-of-the-art string kernel [8], which we shall refer to as *STK17*, and we used an implementation of STK17 downloadable from https://github.com/mufarhan/sequence_class_NIPS_2017. We used LIBLINEAR [7] for training linear SVM with SFMEDM and SFMCGK. We trained nonlinear SVM with GAK, LAK, ESPKernel and CGKKernel using LIBSVM [3]. We performed threefold cross-validation for each dataset and measured

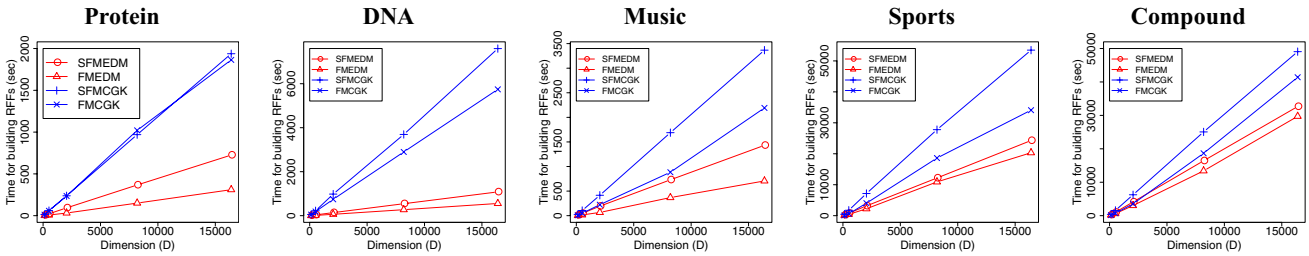

Fig. 3 Time in seconds for building vectors of RFFs for various dimensions D

Table 4 Average error by SFMEDM and FMEDM for each dataset

Method	Protein	DNA	Music	Sports	Compound
SFMEDM($D = 128$)	7.054(± 5.320)	7.051(± 5.318)	7.058(± 5.318)	7.058(± 5.319)	7.057(± 5.317)
FMEDM($D = 128$)	7.055(± 5.321)	7.054(± 2.664)	7.059(± 5.318)	7.059(± 5.320)	7.057(± 5.318)
SFMEDM($D = 512$)	3.523(± 2.662)	3.526(± 2.663)	3.526(± 2.662)	3.525(± 2.662)	3.527(± 2.662)
FMEDM($D = 512$)	3.526(± 2.665)	3.526(± 2.666)	3.526(± 2.663)	3.526(± 2.663)	3.526(± 2.662)
SFMEDM($D = 2048$)	1.762(± 1.332)	1.762(± 1.332)	1.762(± 1.332)	1.762(± 1.332)	1.761(± 1.331)
FMEDM($D = 2048$)	1.763(± 1.332)	1.762(± 1.332)	1.762(± 1.331)	1.763(± 1.331)	1.762(± 1.331)
SFMEDM($D = 8192$)	0.881(± 0.666)	0.881(± 0.665)	0.879(± 0.665)	0.881(± 0.665)	0.876(± 0.664)
FMEDM($D = 8192$)	0.880(± 0.666)	0.881(± 0.666)	0.881(± 0.666)	0.881(± 0.666)	0.881(± 0.665)
SFMEDM($D = 16{,}384$)	0.623(± 0.471)	0.623(± 0.470)	0.621(± 0.470)	0.623(± 0.470)	0.606(± 0.461)
FMEDM($D = 16{,}384$)	0.628(± 0.470)	0.623(± 0.471)	0.623(± 0.471)	0.623(± 0.471)	0.623(± 0.471)

All values are multiplied by 10^2

Table 5 Average error by SFMCGK and FMCGK for each dataset

Method	Protein	DNA	Music	Sports	Compound
SFMCGK($D = 128$)	7.056(± 5.319)	7.051(± 5.318)	7.059(± 5.320)	7.057(± 5.319)	7.056(± 5.317)
FMCGK($D = 128$)	7.054(± 5.316)	7.055(± 5.322)	7.059(± 5.319)	7.057(± 5.319)	7.060(± 5.319)
SFMCGK($D = 512$)	3.524(± 2.662)	3.526(± 2.663)	3.526(± 2.662)	3.525(± 2.662)	3.525(± 2.661)
FMCGK($D = 512$)	3.523(± 2.664)	3.526(± 2.664)	3.527(± 2.661)	3.525(± 2.662)	3.527(± 2.663)
SFMCGK($D = 2048$)	1.761(± 1.331)	1.762(± 1.332)	1.763(± 1.332)	1.763(± 1.332)	1.762(± 1.331)
FMCGK($D = 2048$)	1.762(± 1.332)	1.761(± 1.331)	1.332(± 1.763)	1.762(± 1.331)	1.763(± 1.331)
SFMCGK($D = 8192$)	0.881(± 0.662)	0.881(± 0.665)	0.881(± 0.665)	0.881(± 0.665)	0.869(± 0.663)
FMCGK($D = 8192$)	0.881(± 0.666)	0.881(± 0.666)	0.881(± 0.666)	0.881(± 0.665)	0.881(± 0.666)
SFMCGK($D = 16{,}384$)	0.623(± 0.471)	0.623(± 0.470)	0.632(± 0.470)	0.623(± 0.470)	0.589(± 0.453)
FMCGK($D = 16{,}384$)	0.623(± 0.471)	0.623(± 0.470)	0.632(± 0.471)	0.623(± 0.470)	0.623(± 0.470)

All values are multiplied by 10^2

the prediction accuracy by *the area under the ROC curve (AUC)*. Dimension D of the vectors of RFFs and D2KE was examined for $D = \{128, 512, 2048, 8192, 16{,}384\}$. We selected the best parameter achieving the highest AUC among all combinations of the kernel's parameter $\beta = \{1, 10, 100, 1000, 10000\}$ and the SVM's parameter $C = \{0.001, 0.01, 0.1, 1, 10, 100\}$.

Table 6 shows the execution time for building RFFs and computing kernel matrices in addition to training linear/nonlinear SVM for each method. LAK was applied to only "Protein" because its scoring function was optimized for protein sequences. It took 9 h for LAK to finish the execution, which was the most time-consuming of all the methods in the case of "Protein." The execution of GAK finished within 48 h for "Protein" and "Music" only, and it took around 7 h and 28 h for "Protein" and "Music," respectively. The executions of D2KE did not finish within 48 h for three large datasets of "Music," "Sports" and "Compound." In addition, the executions of EDMKernel and CGKKernel did not finish within 48 h for "Sports" and "Compound." These results suggest that existing alignment kernels are unsuitable for applications to massive string datasets. The executions of D2KE did not finish when large dimensions (e.g., $D = 8192$ and $D = 16{,}384$) were used, which showed that creating high-dimensional vectors for achieving high classification accuracies by D2KE is time-consuming. The executions of SFMEDM and SFMCGK finished with 48 h for all datasets. SFMEDM and SFMCGK took around 9 h and 13 h, respectively, for "Compound" consisting of 1.3 million strings in the setting of large $D = 16{,}382$.

Table 6 Execution time in seconds for building feature vectors and computing kernel matrices in addition to training linear/nonlinear SVM for each method

Method	Protein	DNA	Music	Sports	Compound
SFMEDM(D = 128)	5	8	11	204	261
SFMEDM(D = 512)	22	34	47	799	1022
SFMEDM(D = 2048)	93	138	193	3149	4101
SFMEDM(D = 8192)	367	544	729	12,179	16,425
SFMEDM(D = 16,384)	725	1081	1430	24,282	32,651
SFMCGK(D = 128)	14	52	26	452	397
SFMCGK(D = 512)	60	222	104	1747	1570
SFMCGK(D = 2048)	237	981	415	7156	6252
SFMCGK(D = 8192)	969	3693	1688	27,790	25,054
SFMCGK(D = 16,384)	1937	7596	3366	53,482	49,060
D2KE($D = 128$)	319	4536	296	8139	1641
D2KE($D = 512$)	1250	19,359	1244	34,827	6869
D2KE($D = 2048$)	5213	76,937	5018	140,187	28,116
D2KE($D = 8192$)	21,208	> 48 h	19,716	> 48 h	> 48 h
D2KE(D = 16,384)	43,417	> 48 h	38,799	> 48 h	> 48 h
LAK	31,718	–	–	–	–
GAK	25,252	> 48 h	101,079	> 48 h	> 48 h
EDMKernel	20	28	162	> 48 h	> 48 h
STK17	3218	917	> 48 h	> 48 h	> 48 h

Figure 4 shows amounts of memory consumed for training linear/nonlinear SVM for each method, where GAK, LAK, EDMKernel, CGKKernel and STK17 are represented as "Kernel." "Kernel" required a small amount of memory for the small datasets (namely "Protein," "DNA" and "Music"), but it required a huge amount of memory for the large datasets (namely "Sports" and "Compound"). For example, it consumed 654 GB and 1.3 TB of memory for "Sports" and "Compound," respectively. The memories for SFMEDM, SFMCGK and D2KE were at least one order of magnitude smaller than those for "Kernel." SFMEDM, SFMCGK and D2KE required 36 GB and 166 GB of memory for "Sports" and "Compound" in the case of large $D = 16{,}382$, respectively. These results demonstrated the high memory efficiency of SFMEDM and SFMCGK. Although training linear SVM with vectors built by D2KE was space-efficient, prediction accuracies were not high, which is presented next.

Figure 5 shows the classification accuracy of each method, where the results for the methods not finished with 48 h were not plotted. The prediction accuracies of SFMEDM and SFMCGK were improved for larger D. The prediction accuracy of SFMEDM was higher than that of SFMCGK for any D on all datasets and was also higher than those of all the kernel methods (namely LAK, GAK,

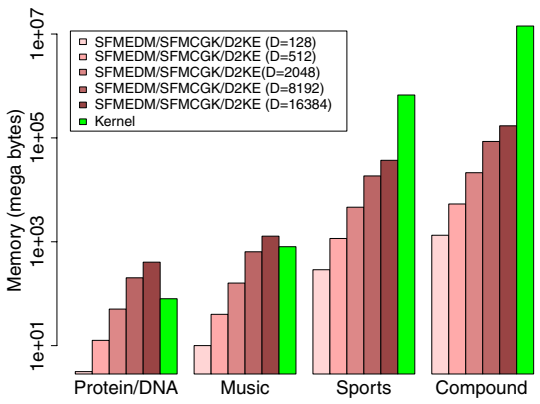

Fig. 4 Memory in mega bytes for training SVM for each method. "Kernel" represents GAK, LAK, EDMKernel, CGKKernel and STK17

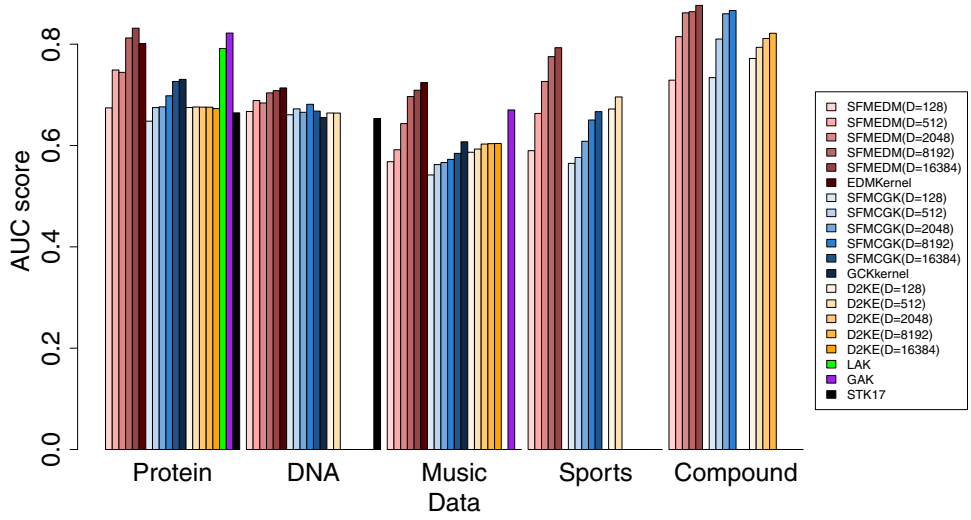

Fig. 5 AUC score for each method

ESPKernel, CGKKernel and STK17). The prediction accuracies of D2KE were worse than those of SFMEDM and were not improved for even large D. One possible reason for those results is that EDM used by SFMEDM includes a substring move operation in addition to typical string operations and it is more efficient string distance than other string distances (e.g., edit distance). These results suggest that SFMEDM can achieve the highest classification accuracy and it is much more efficient than the other methods in terms of memory and time for building RFFs and training SVM.

8 Conclusion

We have presented the first feature maps for alignment kernels, which we call SFMEDM, presented SFMs for computing RFFs space-efficiently and demonstrated its ability to learn SVM for large-scale string classifications with various massive string data, and we demonstrate the superior performance of SFMEDM with respect to prediction accuracy, scalability and computation efficiency. Our SFMEDM has the following appealing properties:

(1) *Scalability* SFMEDM is applicable to massive string data (see Sect. 7).
(2) *Fast Training* SFMEDM trains SVMs fast (see Sect. 7.3).
(3) *Space Efficiency* SFMEDM trains SVMs space-efficiently (see Sect. 7.3).
(4) *Prediction Accuracy* SFMEDM can achieve high prediction accuracy (see Sect. 7.3).

SFMEDM opens the door to new application domains such as bioinformatics and natural language processing, in which large-scale string processing with kernel methods was too restrictive so far.

An idea behind SFMEDM is general and is applicable to other data types such as tree and graph. Thus, an important future work is to apply SFM to other structure kernels such as tree kernels and graph kernels for enhancing a scalability of kernel methods.

Appendix

Edit-Sensitive Parsing

In the next section, we introduce *left preferential parsing (LPP)* as a basic algorithm of ESP. In the later part of this section, we present the ESP algorithm.

(1) Left Preferential Parsing (LPP)

The key idea of LPP is to make pairs of nodes from the left to the right positions preferentially in a sequence of nodes at an ESP tree and make triples of the remaining three nodes. Then, ESP builds type-2 nodes for these pairs of nodes and a type-1 node for the triple of nodes. In this way, LPP builds an ESP tree in a bottom-up manner.

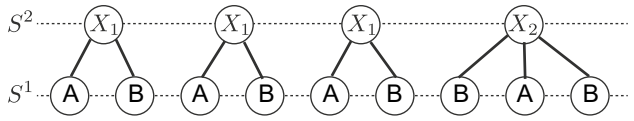

Fig. 6 Example of LPP for a sequence $S' = ABABABBAB$ with odd length

More precisely, if the length of sequence S^ℓ at the ℓth level of the ESP tree is even, LPP makes pairs of $S^\ell[2i-1]$ and $S^\ell[2i]$ for all $i \in [1, |S^\ell|/2]$ and builds type-2 nodes for all the pairs. Thus, $S^{\ell+1}$ at the $(\ell+1)$th level of the ESP tree is a sequence of type-2 nodes. If the length of sequence $S^{\ell+1}$ of the $(\ell+1)$th level of the ESP tree is odd, LPP makes pairs of $S^\ell[2i-1]$ and $S^\ell[2i]$ for each $i \in [1, |S^\ell/2-3|]$, and it makes triple of $S^\ell[2i-2]$, $S^\ell[2i-1]$ and $S^\ell[2i]$. LPP builds type-2 nodes for pairs of nodes and type-1 node for the triple of nodes. Thus, $S^{\ell+1}$ at the $(\ell+1)$th level of the ESP tree is a sequence of type-2 nodes (except the last node) and a type-1 node as the last node. LPP builds an ESP tree in a bottom-up manner; that is, it builds an ESP tree from leaves (i.e., $\ell = 1$) to the root. See Fig. 6 for an example of this ESP-tree building.

A crucial drawback of LPP is that it builds completely different ESP trees even for similar strings. For example, as shown in Fig. 1, $S' = AABABABBAB$ is a string where character A is inserted at the first position of S. Although S' and S are similar strings, LPP builds completely different ESP trees, namely T' and T for S' and S, respectively, resulting in a large difference between EDM $EDM(S', S)$ and L_1 distance $||V(S') - V(S)||_1$ for characteristic vectors V' and V. Thus, LPP lacks the ability to approximate EDM.

The ESP Algorithm

ESP uses an engineered strategy while using LPP in its algorithm. ESP classifies a string into substrings of three categories and applies different parsing strategies according to those categories. An ESP tree for an input string is built by gradually applying this parsing strategy of ESP to strings from the lowest to the highest level of the tree.

Given sequence S^ℓ, ESP divides S^ℓ into subsequences in the following three categories: (1) a substring such that all pairs of adjacent node labels are different and substring length is at least 5. Formally, a substring starting from position s and ending at position e in S satisfies $S^\ell[i] \neq S^\ell[i+1]$ for any $i \in [s, e-1]$ and $(e-s+1) \geq 5$; (2) a substring of the same node label and with length of at least 5. Formally, a substring starting from position s and ending at position e satisfies $S^\ell[i] = S^\ell[i+1]$ for any $i \in [s, e-1]$ and $(e-s+1) \geq 5$; (3) neither of categories (1) and (2).

landmarks			✓		✓			✓
labels	-	1	0	1	0	2	4	1
binary	000	001	110	111	010	100	000	101
sequence	A	B	G	H	C	E	A	F

Fig. 7 Example of alphabet reduction

After classifying a sequence into subsequences of the above three categories, ESP applies different parsing methods to each substring according to their categories. ESP applies LPP to each subsequence of sequence S^ℓ in categories (2) and (3), and it builds nodes at $(\ell+1)$-level. For subsequences in category (1), ESP applies a special parsing technique named *alphabet reduction*.

Alphabet Reduction alphabet reduction is a procedure for converting a sequence to a new sequence with alphabet size of 3 at most. For each symbol $S^\ell[i]$, the conversion is performed as follows: $S^\ell[i-1]$ is a left adjacent symbol of $S^\ell[i]$. Suppose $S^\ell[i-1]$ and $S^\ell[i]$ are represented as binary integers. Let p be the index of the least-significant bit in which $S^\ell[i-1]$ differs from $S^\ell[i]$, and let $bit(p, S^\ell[i])$ be the binary integer of $S^\ell[i]$ at the pth bit index. label $label(S^\ell[i])$ is defined as $2p + bit(p, S^\ell[i])$ and $label(S^\ell[i])$ is computed for each position i in S^ℓ. When this conversion is applied to a sequence of alphabet Σ, the alphabet size of the resulting label sequence is $2\log|\Sigma|$, In addition, an important property of labels is that all adjacent labels in a label sequence are different, i.e., $label(S^\ell[i]) \neq label(S^\ell[i-1])$ for all $i \in [2, |S^\ell|]$. Thus, this conversion can be iteratively applied to a new label sequence, namely $label(S^\ell[1])label(S^\ell[2]) \ldots label(S^\ell[L])$, until its alphabet size is at most 6.

The alphabet size is reduced from $\{0, 1, \ldots, 5\}$ to $\{0, 1, 2\}$ as follows. First, each 3 in a sequence is replaced with the least element from $\{0, 1, 2\}$ that does not neighbor the 3. Then, the same procedure is repeated for each 4 and 5, which generates a new sequence (A) of node labels drawn from $\{0, 1, 2\}$, where no adjacent characters are identical.

Any position i that is a *local maximum*, i.e., $A[i-1] < A[i] > A[i+1]$, is then selected. Those positions are called *landmarks*. In addition, any position i that is a *local minimum*, i.e., $A[i-1] > A[i] < A[i+1]$, and not adjacent to an already chosen landmark, is selected as a landmark. An important property for those landmarks is that for any two successive landmark positions, i and j, either $|i-j| = 2$ or $|i-j| = 3$ hold, because A is a sequence of no adjacent characters in alphabet $\{0, 1, 2\}$. Alphabet reduction for sequence $ABGHCEAF$ is illustrated in Fig. 7.

Finally, type-2 nodes (respectively, type-3 nodes) are built for subsequences between landmarks i and j of length $|i-j| = 2$ (respectively, $|i-j| = 3$).

The computation time of ESP is $O(NL)$.

References

1. Bahlmann C, Haasdonk B, Burkhardt H (2002) Online handwriting recognition with support vector machines-a kernel approach. In: Proceedings of the 8th international workshop on frontiers in handwriting recognition, pp 49–54
2. Chakraborty D, Goldenberg E, Koucký M (2016) Streaming algorithms for embedding and computing edit distance in the low distance regime. In: Proceedings of the 48th annual ACM symposium on theory of computing, pp 715–725
3. Chang CC, Lin CJ (2011) LIBSVM: a library for support vector machines. ACM Trans Intell Syst Technol 2:27:1–27:27
4. Cormode G, Muthukrishnan S (2007) The string edit distance matching problem with moves. ACM Trans Algorithms 3:2:1–2:19
5. Cuturi M (2011) Fast global alignment kernels. In: Proceedings of the 28th international conference on machine learning, pp 929–936
6. Cuturi M, Vert JP, Birkenes O, Matsui T (2007) A kernel for time series based on global alignments. In: Proceedings of the 2007 IEEE international conference on acoustics, speech and signal processing, pp 413–416
7. Fan RE, Chang KW, Hsieh CJ, Wang XR, Lin CJ (2008) LIBLINEAR: a library for large linear classification. J Mach Learn Res 9:1871–1874
8. Farhan M, Tariq J, Zaman A, Shabbir M, Khan I (2017) Efficient approximation algorithms for strings kernel based sequence classification. In: Proceedings of the 30th international conference on neural information processing systems, pp 6938–6948
9. Ferris MC, Munson TS (2003) Interior-point methods for massive support vector machines. SIAM J Optim 13:783–804
10. Gärtner T (2003) A survey of kernels for structured data. ACM SIGKDD Explor Newslett 5:49–58
11. He R, McAuley J (2016) Ups and downs: modeling the visual evolution of fashion trends with one-class collaborative filtering. In: Proceedings of the 25th international world wide web conference, pp 507–517
12. Hofmann T, Schölkopf B, Smola AJ (2008) Kernel methods in machine learning. Anna Stat 36:1171–1220
13. Joachims T (2006) Training linear SVMs in linear time. In: Proceedings of the 12th ACM conference on knowledge discovery and data mining, pp 217–226
14. Kanehisa M, Furumichi M, Tanabe M, Sato Y, Morishima K (2017) Kegg: new perspectives on genomes, pathways, diseases and drugs. Nucleic Acids Res 45:D353–D361
15. Kim S, Thiessen PA, Bolton EE, Chen J, Fu G, Gindulyte A, Han L, He J, He S, Shoemaker BA, Wang J, Yu B, Zhang J, Bryant SH (2016) Pubchem substance and compound databases. Nucleic Acids Res 44:D1202–D1213
16. Kuksa PP, Huang PH, Pavlovic V (2008) Kernel methods and algorithms for general sequence analysis. Rutgers computer science technical report, vol. RU-DCS-TR630
17. Le Q, Sarlos T, Smola A (2013) Fastfood—approximating kernel expansions in loglinear time. In: Proceedings of the 30th international conference on machine learning, pp 244–252
18. Leslie C, Eskin E, Noble WS (2002) The spectrum kernel: a string kernel for SVM protein classification. In: Proceedings of the 7th pacific symposium on biocomputing, pp 566–575
19. Li P (2017) Linearized GMM kernels and normalized random Fourier features. In: Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining, pp 315–324
20. Li P, Shrivastava A, Moore J, Köning AC (2011) Hashing algorithms for large-scale learning. In: Advances in neural information processing systems, pp 2672–2680
21. Lodhi H, Saunders C, Shawe-Taylor J, Cristianini N, Watkins C (2002) Text classification using string kernels. J Mach Learn Res 2:419–444
22. Maruyama S, Tabei Y, Sakamoto H, Sadakane K (2013) Fully-online grammar compression. In: Proceedings of international symposium on string processing and information retrieval, pp 218–229
23. McAuley J, Targett C, Shi J, van den Hagel A (2015) Image-based recommendations on styles and substitutes. In: Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval, pp 43–52
24. Pham N, Pagh R (2013) Fast and scalable polynomial kernels via explicit feature maps. In: Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining, pp 239–247
25. Rahimi A, Recht B (2007) Random features for large-scale kernel machines. In: Advances in neural information processing systems, pp 1177–1184
26. Saigo H, Vert JP, Ueda N, Akutsu T (2004) Protein homology detection using string alignment kernels. Bioinformatics 20:1682–1689
27. Shapira D, Storer JA (2002) Edit distance with move operations. In: Proceedings of the 13th symposium on combinatorial pattern matching, vol 2373, pp 85–98
28. Shimodaira H, Noma K, Nakai M, Sagayama S (2001) Dynamic time-alignment kernel in support vector machine. In: Advances in neural information processing systems, pp 921–928
29. Singh R, Sekhon A, Kowsari K, Lanchantin J, Wang B, Qi Y (2017) Gakco: a fast gapped k-mer string kernel using counting. In: Proceedings of the European conference on machine learning and principles and practice on knowledge discovery in databases
30. Smith TF, Waterman MS (1981) Identification of common molecular subsequences. J Mol Biol 147:195–197
31. Takabatake Y, Tabei Y, Sakamoto H (2014) Improved ESP-index: a practical self-index for highly repetitive texts. In: Proceedings of the 13th international symposium on experimental algorithms, pp 338–350
32. Wu L, Yen IE, Xu F, Ravikumar P, Witbrock M (2018) D2KE: from distance to kernel and embedding. CoRR arXiv:1802.04956
33. Wu L, Yen IE, Yi J, Xu F, Lei Q, Witbrock M (2018) Random warping series: a random features method for time-series embedding. In: Proceedings of the 21st international conference on artificial intelligence and statistics, pp 793–802
34. Yu F X, Suresh AT, Choromanski KM, Holtmann-Rice DN, Kumar S (2016) Orthogonal random features. In: Proceedings of the 29th international conference on neural information processing systems, pp 1975–1983
35. Zhang H, Zhang Q (2017) EmbedJoin: efficient edit similarity joins via embeddings. In: Proceedings of the 23rd SIGKDD international conference on knowledge discovery and data mining, pp 585–594
36. Zhou F, De la Torre F, Cohn JF (2010) Unsupervised discovery of facial events. In: Proceedings of the 2010 IEEE computer society conference on computer vision and pattern recognition, pp 2574–2581